Advanced Higher Computing Assignment

"Meteor Rush" - Arcade Style Game

Scott Jarvis

08/11/16 – 21/04/17

Contents

Project Proposal

Project Aim:

Side scrolling games are common across the computers of the world, being generally easy to learn (and to create), but difficult to create or play *well*. For my advanced higher final computing project, I elected to make a basic arcade-style side scrolling shooter that should ideally be enjoyable and moderately challenging. As well as this, such a game can easily incorporate many of the features required for an advanced higher project, such as the use of complex searches and sorts, multiple-layered arrays, and so on.

End user group:

Anyone who is willing to play the game and has rudimentary computing skills (i.e. moving a mouse, pressing keys).

Main Features of the Project:

- The program should have an attractive design for background & sprites
- The user should be able to exit the game at any time
- The program should continue until they hit an obstacle
- The program should keep track of a player's score, and record it when done
- The program should store 40 user names and high-scores
- The program should be able to display the best 10 times in descending order
- The program should have multiple states (title screen, score, gameplay, gameover, etc.)
- The program should allow the user to replay until they decide to quit
- The program should allow the user to search the best scores for their name and display them
- The program should display the current score, boost fuel and time while playing
- The program should allow the user to view a help screen when necessary
- The program should provide feedback that is easy to understand
- The program should provide the user with audio feedback

AH Skills:

- Interface that can validate input
- File interfacing
- Python operations

Hardware, Software and Resources Needed:

To complete this program I will require a computer that is capable of running and has the chosen Software Development application (Python & Pygame) installed. A bank of resource images will also be required for the program. The computer will also require a graphical application, such as the free software "paint.net" to make edits to the images in question. The application will be used to resize and edit images.

Feasibility Issues:

Economic:

- This project will have no economic issues, as the plan, analysis, implementation and testing of the project can all be done without a cost, as all software in use is free to use or provided by the school.

Legal:

- This project should be legally feasible: no personal data is stored or even asked for, meaning that the Data Protection Act is not applicable to the game. The Copyright, Design and Patents Act may have some effects on the project however, as the game will use multiple sources for images, mostly from Google's image browser. This should be fine, however, as the game is not being distributed beyond use as an Advanced Higher Computing project. The music in use in the game will be stock music if possible, but will be songs I have provided as placeholders if this is not an option.

Technical:

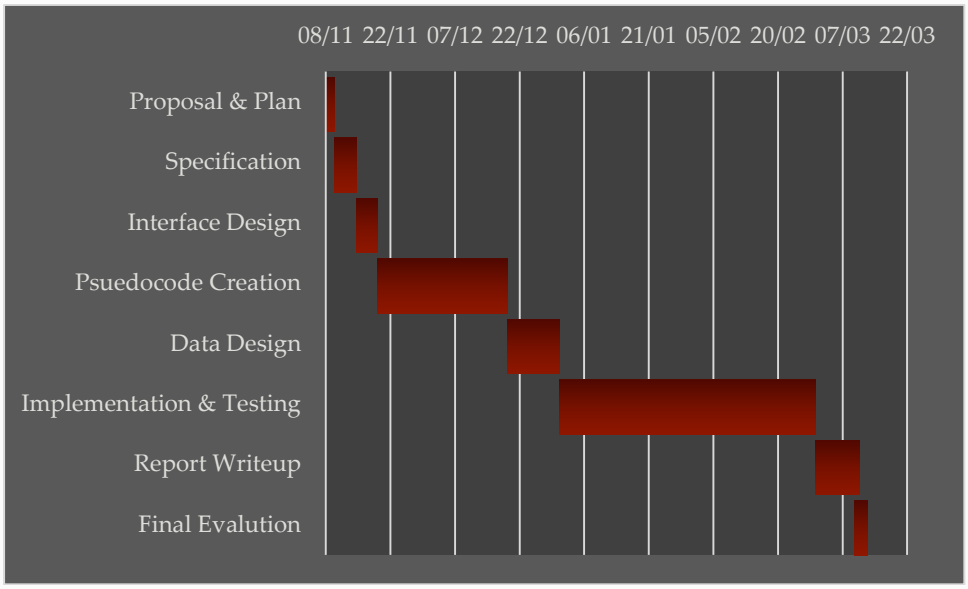
- This project should be technically feasible, as no software beyond python, a few modules for it, and an interpreter is necessary. I may need to make use of online forums to gain additional advice on how to implement certain features into the project, however.

Time:

- This project must be completed within 50 or so hours, and in order to keep to this timescale, any over-ambitious features of the project may need to be cut.

Plan

A Gantt chart has been drafted on time constraints within the project:



Stage	Estimated Time
➢ Analysis	3 Hours
➢ Design	10 Hours
➢ Implementation & Testing	28 Hours
➢ Documentation	7 Hours
➢ Evaluation	3 Hours

Possible Solutions

The program could be made in any programming language with a functional visual aspect to it, such as java, php, or python (with modules attached). For simplicity's sake, I have chosen to program the game in python, as it is the language with which I am most familiar. The "PyGame" module will be in use by the program, to give it functional input and output methods such as keyboard input and moving, interactive windows. The easiest control scheme to implement is one based on the movement and clicking of a mouse, as mice are fairly universal at this point, and intuitive for a user to move and use.

The likeliest end user for the project is, as was stated above, basically anybody who is willing to play the game. To this end, I drafted an end-user survey for use with various pupils across the Higher and Advanced Higher computing class. The survey is provided below, as well as some example responses.

Project User Survey – Side Scrolling Shooter

I am creating a small python-based side scrolling game for anyone to play. The game will include basic features, such as a high-score system and background scrolling, and will be set in a space environment.

Please answer the below questions as best you can as a potential end user of the game, so that I can design it accordingly.

1) What name would you prefer for the game?

    a. Star Survival
    b. Meteor Rush
    c. The Space Race
    d. Other (please specify):

2) What would you like the vehicle that the player can control to be?

3) Would you prefer mouse–based controls or keyboard-based controls?

    a. Mouse
    b. Keyboard

4) Would you prefer the game have sound effects, have background music, neither or both?

    a. Sound Effects
    b. Background Music
    c. Neither
    d. Both

5) Do you have any other suggestions?

Please provide some basic information about yourself, so that I know what user group I'm designing for. If you wish to remain anonymous, feel free not to fill anything in:

Gender: Year Group:

Project User Survey – Side Scrolling Shooter

I am creating a small python-based side scrolling game for anyone to play. The game will include basic features, such as a high-score system and background scrolling, and will be set in a space environment.

Please answer the below questions as best you can as a potential end user of the game, so that I can design it accordingly.

1) What name would you prefer for the game?

    a. Star Survival
    b. Meteor Rush
    c. The Space Race x
    d. Other (please specify):

2) What would you like the vehicle that the player can control to be?

Rocket / UFO

3) Would you prefer mouse–based controls or keyboard-based controls?

    a. Mouse x
    b. Keyboard

4) Would you prefer the game have sound effects, have background music, neither or both?

    a. Sound Effects
    b. Background Music
    c. Neither
    d. Both x

5) Do you have any other suggestions?

Escalating difficulty?

Please provide some basic information about yourself, so that I know what user group I'm designing for. If you wish to remain anonymous, feel free not to fill anything in:

Gender: Male Year Group: 6th

Project User Survey – Side Scrolling Shooter

I am creating a small python-based side scrolling game for anyone to play. The game will include basic features, such as a high-score system and background scrolling, and will be set in a space environment.

Please answer the below questions as best you can as a potential end user of the game, so that I can design it accordingly.

1) What name would you prefer for the game?

    a. Star Survival
    b. Meteor Rush
    c. The Space Race
    d. Other (please specify):

2) What would you like the vehicle that the player can control to be?

A spaceship

3) Would you prefer mouse–based controls or keyboard-based controls?

    a. Mouse
    b. Keyboard

4) Would you prefer the game have sound effects, have background music, neither or both?

    a. Sound Effects
    b. Background Music
    c. Neither
    d. Both

5) Do you have any other suggestions?

Make sure it has a classic three character high score screen thing.

Please provide some basic information about yourself, so that I know what user group I'm designing for. If you wish to remain anonymous, feel free not to fill anything in:

Gender: Female Year Group: 6th

Requirements Specification

Purpose

The purpose of this project is to create a small, easy to play game to be played in order to generate a score, which will then be added to a file, sorted amongst the information already on the file, and then shown to the user in the form of the top three scores with related names.

Scope & Boundaries

The project will be able to have one user playing at any single time, and will be able to hold a technically unlimited range of scores. The file itself will grow as more are added, however, so a max file size will eventually be reached. The score itself is also technically unlimited, although the program will increase the rate at which meteors are created as the game progresses, and eventually become unplayable. Some features (buttons, for instance) may need to be cut or modified as the development of the game progresses to make it actually creatable within the boundaries (particularly time based) of Advanced Higher Computing Science.

User Requirements

Unless the project is compiled into an executable file, the user will require python and all of the modules in use in the code to be able to run the program. The python version in question is 3.0, the latest at the time of writing. The modules in use are pygame, random, time, and csv. A user will need a functioning mouse, keyboard, and monitor, in addition to a standard computer running windows, and storage containing the program.

Functional Requirements

In order to create the program, an IDE installation will be required. The IDE I have chosen to write the program with python in is Wing, a popular paid-for IDE which has a free-to-use version available with slightly reduced functionality, as it provides many useful features and an easy-to-understand UI.

Inputs & Outputs

The program will take in input from the user's mouse and keyboard, a comma separated value file used to store the scores generated by the game, and a large selection of graphics (mostly png files) and audio files (ogg files) for the game to render/play while in use.

The program will output visually in the form of rendered graphics and text in a window on the user's screen, as well as audio output via the music files. The program will also output newly modified information to the same csv file storing the scores, to update the file.

Test Plan

Subject	Input	Expected Output
“MousePos” value used in Rocket class’s “moveIt” function	Mouse moved upwards	“cursor” graphic goes upward to mouse position, “x” value is lowered
^	Mouse moved left	“cursor” graphic goes left to mouse position, “y” value is lowered
Bubble sort function	Csv file containing “TOP,999 LOW,0 MID,500 ESC,123”	Csv file is sorted into “TOP,999 MID,500 ESC,123 LOW,0”
Bubble sort function on a second iteration	Csv file containing “TOP,999 MID,500 ESC,123 LOW,0”	Csv file remains unchanged
“Boosting” function	Player uses left click	Screen moves past at an increased rate, slows back to normal when clicking stops.
Player death	Cursor moves position to that of a “meteor”	Rocket is removed, explosion graphic is placed, worldState changes to “game over” state
“Laser fire” function	Player clicks right mouse button	“Laser” entity is created on cursor location, begins moving right.
Name input	Worldstate is 4, player enters three letters to make name	playerName variable is created by concatenation of three letters, game progresses to next state.

User Interface

The following section will describe the Graphical User Interface (GUI) of the program in each of the "world states" of the project. The different states are used as a simple sorting system for each screen, each providing a different function. The separate states are Main Menu (0), Gameplay (1), Game Over (2), Help (3), and High Scores (4).

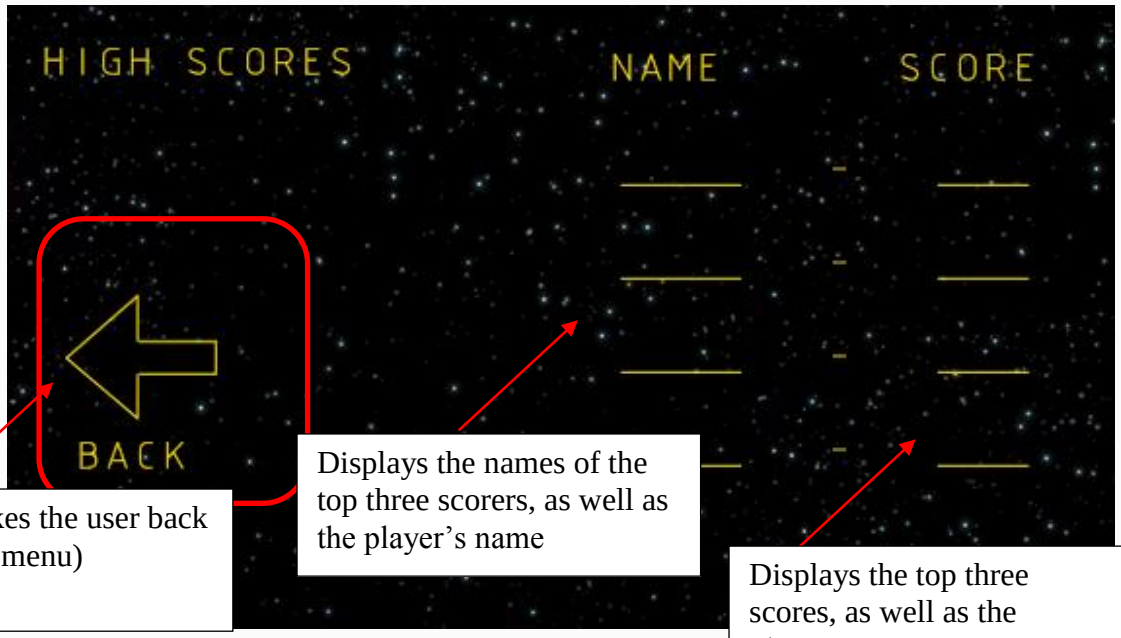
Main Menu:

The Main Menu Interface will contain a small intro to the game, an easy to understand layout and buttons that link to the other screens in the program. These buttons simply change the worldstate variable to whatever is required. As well as this, the main menu will have a logo, a title, and the scrolling space based background that can be found across the program.

Buttons. If the user clicks on these, the worldState variable changes. The choice of buttons leaves the UI as simple, accessible, and easy to understand.

High Scores:

The High Scores page keeps the same general aesthetic design as the main menu, but displays the top 3 scores and the player's score instead of the game title and buttons.



Button (Takes the user back to the main menu)

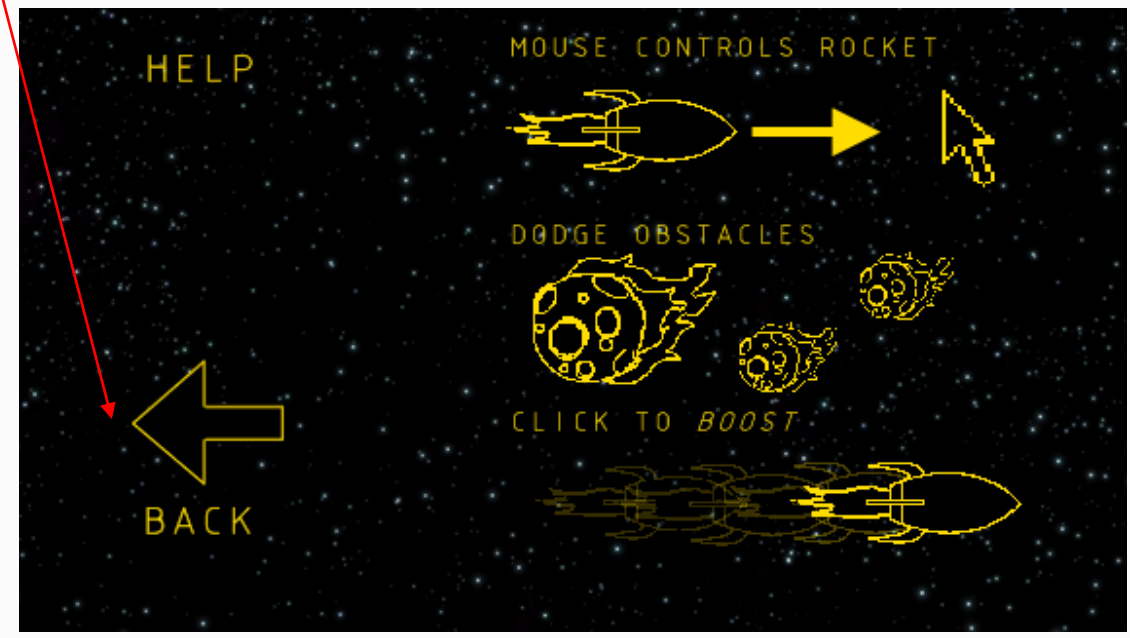
Displays the names of the top three scorers, as well as the player's name

Displays the top three scores, as well as the player's score.

Other high scores will be available from the csv file storing the other scores.

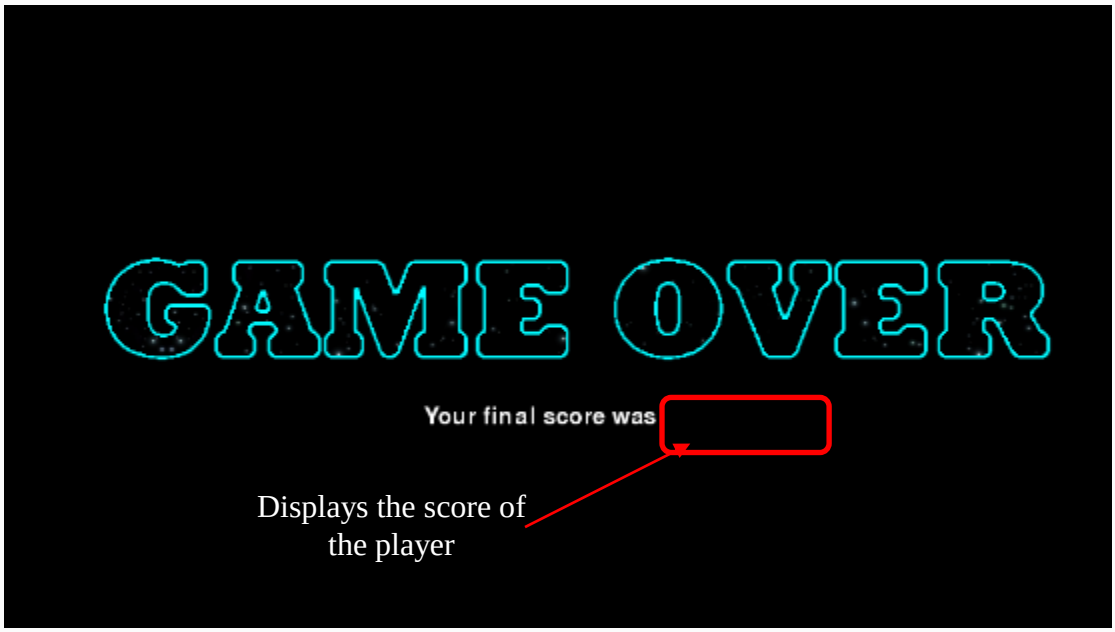
Help:

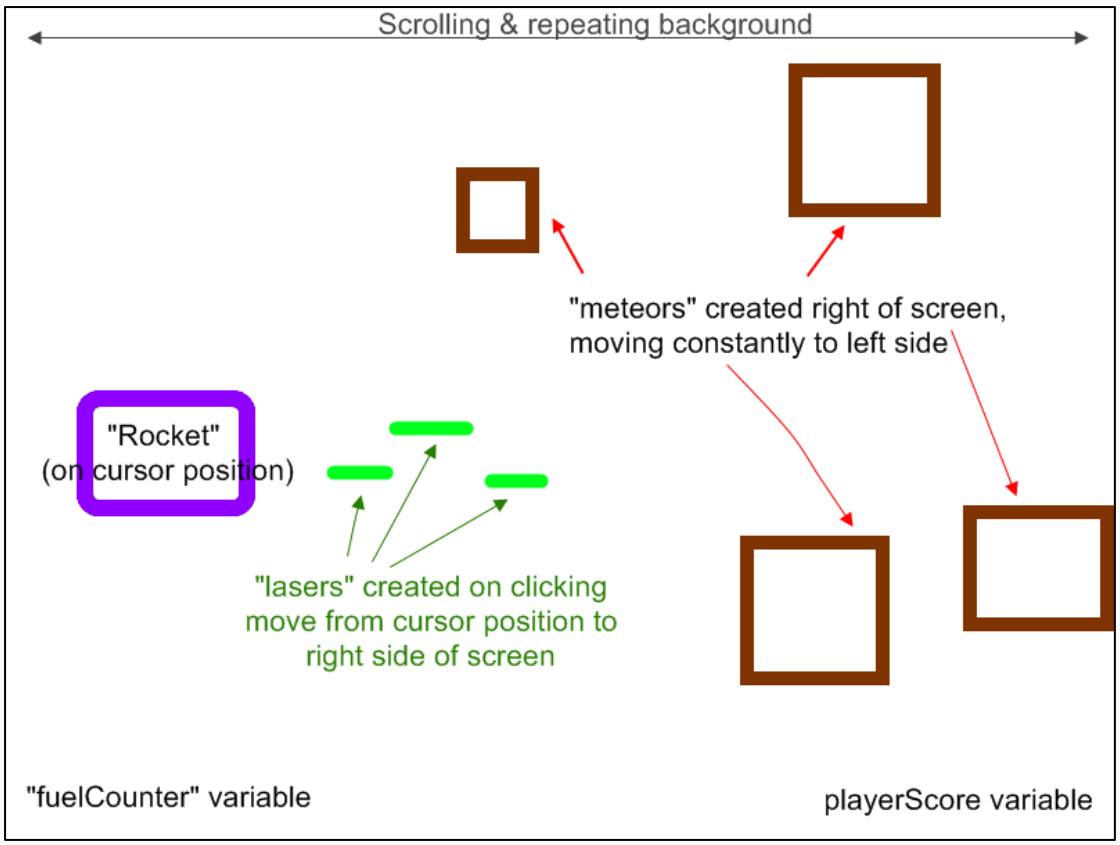
The help screen will be fairly simple, and just display how the game works, with accompanying graphics.



Game Over:

The game over screen is going to be fairly straightforward, utilizing a mostly solid black background with text spelling “Game Over” in the centre of the screen. The letters themselves will be transparent with thick outlines, allowing the scrolling background to be visible within the text. As well as this, a notification of the final score will be provided, which itself will be calculated by combining multiple figures, including time survived, distance travelled, and so on.





Gameplay

Finally, the gameplay section is the most important section of the game, being what makes it more than just a glorified number sorter. This will have displays for the current values of the “fuel” counter (used to “boost”), the “score”, and the “time taken”. The core gameplay will consist of a central sprite following the position of the mouse, and randomly created obstacles coming in from the right side of the screen. The user will need to move the mouse and avoid the obstacles, as hitting one of them will trigger a game over. The longer a user survives for, the greater their score will be. If the user triggers a “boost”, the obstacles will move much faster, but the score will increase faster as well. Finally, the user will be able to click with the right mouse button to “shoot” a small laser in front of them. This will move right across the screen, and if it makes contact with an obstacle, the obstacle will be removed.

Pseudocode

Pre-Game

1. Import modules

2. Create classes*

3. Initiate pygame

4. Set up base environment

    4.1. Set dimensions of window
    4.2. Set base colours for graphics
    4.3. Start clock
    4.4. Make font for text displays

5. Set up variables, groups & arrays
6. Create “Display Cards” for each world state
7. Set up functions

    7.1. “explode” function
        7.1.1. create instance of “explosion” object
        7.1.2. add object to explosions group
        7.1.3. load & play sound effect

    7.2. “takeName” function
        7.2.1. Start while loop
            7.2.1.1. Use “keyPress” function to take in input from keyboard
            7.2.1.2. Assign input to part one, two or three of the name
            7.2.1.3. Concatenates the three parts into one string once all three are done
            7.2.1.4. Sets “playerName” to the string
            7.2.1.5. Ends while loop

    7.3. “makeMeteor” function
        7.3.1. Creates instance of “meteor” object at given coordinates on the screen
        7.3.2. adds meteor to meteors group

    7.4. “findPlayerLocation” function
        7.4.1. sets initial location to 1
        7.4.2. checks if every value in a given list matches the player’s name
        7.4.3. if a value doesn’t match, 1 is added to location variable
        7.4.4. if it does match, location is returned
        7.4.5. location is returned if the entire list is searched

7.5. “Hyperspace” function
    7.5.1. displays the “space” object on screen at given coordinates
    7.5.2. decreases the x value of space by 1.75 * the “baseMult” value
    7.5.3. if the x value of space is less than 1000, sets it to 0
    7.5.4. returns the x value of space

7.6. “playMusic” function
    7.6.1. loads music into the program based on what the current value of worldState is

7.7. “keyPress” function
    7.7.1. starts a while loop
    7.7.2. gets input from keyboard
    7.7.3. sets output to whatever key is pressed if a letter is pressed (a-z), and to * if something else is pressed
    7.7.4. ends while loop when a key is pressed
    7.7.5. returns output

7.8. “fireLaser” function
    7.8.1. creates an instance of the laser object
    7.8.2. adds the instance to the laserblasts group

Main Game Loop

1. Starts while loop
2. If music is not playing, “playMusic” function is triggered
3. Searches for active events
    3.1. If user clicks the “close window” button, ends the loop
    3.2. If user moves the mouse, sets the values of “mousePos” to the location of the cursor
4. If the mouse is beyond the edges of the screen during gameplay, sets the cursor location to just before the edge
5. If the worldState is 1, the cursor is set to invisible and the rocket class’s “moveIt” function is triggered with the values of the mouse position
6. Otherwise, the cursor is set to being visible
7. A series of while and if loops are used changing the game as worldState is changed**
8. Ends while loop
9. Exits pygame.

* Class Information *

1. "DisplayCard" Class
   1.1. "init" function
      1.1.1. Creates the object as a pygame sprite
      1.1.2. sets the image to load over the sprite to one of many in a directory depending on what the "make" value is, changing the colour key for transparency accordingly
      1.1.3. sets the "rect" value of the object

2. "Rocket" class
   2.1. "init" function
      2.1.1. Creates the object as a pygame sprite
      2.1.2. Sets the image to load over the sprite, with colour key
      2.1.3. Sets the dimensions of the object
      2.1.4. sets the rect values of the object

   2.2. "moveIt" function
      2.2.1. Sets the x and y values of the object to just behind those of the cursor position

3. "Explosion" class
   3.1. "init" function
      3.1.1. Creates the object as a pygame sprite
      3.1.2. sets up an array for images
      3.1.3. adds a single frame of a gif file to the array
      3.1.4. loops the above line 13 times
      3.1.5. sets the index variable as 0
      3.1.6. loads an image from the array based on the value of index
      3.1.7. sets up the rect values of the explosion

   3.2. "update" function
      3.2.1. adds 1 to index
      3.2.2. sets dead to false
      3.2.3. loads an image based on index
      3.2.4. when all images are loaded, sets dead to true
      3.2.5. kills the object

4. "Meteor" class
   4.1. "init" function
      4.1.1. Creates the object as a pygame sprite
      4.1.2. Sets the image to load over the sprite, with colour key
      4.1.3. Sets the dimensions of the object
      4.1.4. sets the rect values of the object

   4.2. "moveIt" function
      4.2.1. decreases the x value of the object by the value of "speed"

5. “Laser Bolt” class
    5.1. “init” function
        5.1.1. Creates the object as a pygame sprite
        5.1.2. Sets the image to load over the sprite, with colour key
        5.1.3. Sets the dimensions of the object
        5.1.4. sets the rect values of the object

    5.2. “moveIt” function
        5.2.1. increases the x value of the object by the value of “speed”

** worldState Loops **

1. Loop while worldState value is 0 (Main Menu)
    1.1. Call “Hyperspace” function
    1.2. draw “menu” displaycard
    1.3. Set variables that change during gameplay (time, baseMult, score, etc) to 0 or False
    1.4. Kill all sprites on screen (Ensures that only the background and menu are visible)
    1.5. Flip screen
    1.6. Check for events
    1.7. If the mouse is clicked, changes worldState to 1, resets variables, creates an instance of a rocket object and adds it to a group for cursors

2. Loop while worldState value is 1
    2.1. If right mouse button is pressed and cooldown is 0, calls “fireLaser” function and changes cooldown to 100, otherwise if cooldown is more than 0 reduces cooldown
    2.2. If left mouse button is pressed and boostFuel is more than 0, increase baseMult
    2.3. Display fuel & score
    2.4. If the length of time pygame’s clock has been ticking for is larger than the value of timeTillNextObstacle, spawn a meteor on the right side of the screen and increase timeTillNextObstacle by a random amount
    2.5. Call hyperspace function
    2.6. Draw objects on screen
    2.7. Call explosion’s update function
    2.8. Increase baseMult by 1.1 * baseMult
    2.9. Adds 1 to “realTime” counter
    2.10. Call all moveIt functions
    2.11. Check locations of objects:
        2.11.1. If an obstacle’s x value is less than the edge of the screen, it is killed
        2.11.2. If there is a collision between the cursor and an obstacle, the cursor is killed
        2.11.3. If there is a collision between a laser and an obstacle, the obstacle is killed
        2.11.4. If there are no objects in the “cursors” group, calls the explode function with the x and y values of the cursor’s location

2.12. If there are no objects in the cursors group and three seconds have passed, change worldState to 2

3. Loop while worldState value is 2
   3.1. call hyperspace function
   3.2. draw "end" displaycard
   3.3. show score to user via rendered text
   3.4. check for events
   3.5. if a key is pressed or the mouse is clicked, worldState is changed to 3 and arrays holding names & scores are emptied (if there is anything in them)
4. Loop while worldState value is 3
   4.1. Call hyperspace function
   4.2. Draw "nameEntry" displayCard
   4.3. Call takeName function to get playerName
   4.4. Open scoreList.csv
   4.5. Append information from file onto arrays
   4.6. If playerName is the same as a name already on the file, overwrites the score. Otherwise, creates a new entry for the player with the score.
   4.7. Find position of player on scoreList with "findPlayerLocation" function
   4.8. Set worldState to 4

5. Loop while worldState value is 4
   5.1. Call hyperspace function
   5.2. Draw "highScores" displaycard
   5.3. Use font renders to show the player's score, name and position
   5.4. Use more font renders to show the top three scores (and names)
   5.5. If the player is in first, second or third place, congratulate them with another font render
   5.6. check for events
   5.7. if mouse is clicked, worldState is changed back to 0 and writes scores and names back to fill

Data Structure Design

Global Variables (used in all worldStates)

- isRunning, Boolean
- mousePos, integer array
- score, integer
- baseMult, float
- worldState, integer

Files & Modules

- scoreList.csv (for storing/writing scores)
- background.png (background of game)
- mus0 – mus3.ogg (background music)
- kaboom.ogg (explosion sound effect)
- screen0 – screen3.png (graphics for displayCard)
- meteorBase.png (Graphics for objects ingame)
- rocketBody.png (^)
- laserBolt.png (^)
- pic0 – pic12.gif (frames of explosion)

- meteorRush.py (Main file)

- pygame module
- time module
- random module
- csv module

CSV design

Player Name (string)	Score (Int)
Example	1234

Implementation

```
# Meteor Rush         ##
# Made by Scott Jarvis ##
# Portlethen Academy   ##
# Adv Higher Computing ##
# 24/11/16 to 24/03/17 ##

import pygame, random, time, csv

# Classes
class displayCard(pygame.sprite.Sprite):                    # This class is just for the
title & game over cards that display at the start and end

    def __init__(self,make):
        pygame.sprite.Sprite.__init__(self)
        if make == "titleCard":
            self.image =  pygame.image.load("screencards/itleScreen.png")
            self.image.set_colorkey(blue)
        elif make == "endCard":
            self.image =  pygame.image.load("screencards/gameOver.png")
            self.image.set_colorkey(white)                  # Different colourkey allows
for transparency in the image
        elif make == "scoreCard":
            self.image =  pygame.image.load("screencards/scoreName.png")
            self.image.set_colorkey(black)
        elif make == "highCard":
            self.image =  pygame.image.load("screencards/highScores.png")
            self.image.set_colorkey(blue)

        self.rect = pygame.Rect(0, 0, 1000, 500)


class rocket(pygame.sprite.Sprite):                         #This class is the "rocket"
controlled by the cursor

    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        baseImg = pygame.image.load("entities/rocketBody.png")
        self.image = pygame.Surface([65,25])
        self.image.set_colorkey(blue)
        self.image.blit(baseImg,(0,0))
        self.rect = self.image.get_rect()
        self.rect.x = 0
        self.rect.y = 0
```

```
    def moveIt(self, mousePos):                                #constantly sets the current
position of the rocket to just behind the cursor, as to avoid going off window
        self.rect.x = (mousePos[0] - 40)
        self.rect.y = (mousePos[1] - 10)

class explosion(pygame.sprite.Sprite):

    def __init__(self,x,y,why):
        super(explosion, self).__init__()
        self.images = []

        for count in range(13):
            self.images.append(load_image("explode\pic" + str(count) + ".gif"))     # adds
all of the frames of the explosion in the folder to the array
            count += 1

        self.index = 0
        self.image = self.images[self.index]
        self.rect = pygame.Rect(x - 10, y - 40, 200, 200)



    def update(self):
        '''This method iterates through the elements inside self.images and
        displays the next one each tick. For a slower animation, you may want to
        consider using a timer of some sort so it updates slower.'''
        self.index += 1
        ded = False
        if self.index >= len(self.images):
            ded = True
        if ded != True:
            self.image = self.images[self.index]                    # sets the image to
display for the object to the current image in the index
        else:
            self.kill                                           # kills the object if the frame in
use excedes the number of frames in the index
```

```
class obstacle(pygame.sprite.Sprite):

    def __init__(self,name,x,y):

        pygame.sprite.Sprite.__init__(self)

        baseImg = pygame.image.load("entities\meteorBase.png")


        self.image = pygame.Surface([90,90])

        self.image.set_colorkey(white)
        self.image.blit(baseImg,(0,0))

        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y


    def moveIt(self, speed):                    #constantly moves the meteor to the
left at the rate of "speed"
        self.rect.x -= speed

class laserBolt(pygame.sprite.Sprite):
    def __init__(self,x,y):

        pygame.sprite.Sprite.__init__(self)

        baseImg = pygame.image.load("entities\laserBolt.png")

        self.image = pygame.Surface([25,15])

        self.image.set_colorkey(black)
        self.image.blit(baseImg,(0,0))

        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y

    def moveIt(self, speed):                    #constantly moves the meteor to the
left at the rate of "speed"
        self.rect.x += speed
```

```
## PYGAME INITIALISATION ##


pygame.init()

screen = pygame.display.set_mode([1000,500])
pygame.display.set_caption("")
clock = pygame.time.Clock()
black    = (   0,   0,   0)
white    = ( 255, 255, 255)
blue     = (   0,   0, 255)

font = pygame.font.Font(None, 36)

## GROUP CREATION ##

entities = pygame.sprite.Group()
cursors  = pygame.sprite.Group()
boomboom = pygame.sprite.Group()
cards    = pygame.sprite.Group()
buttons  = pygame.sprite.Group()
blasts   = pygame.sprite.Group()

## VARIABLE SETUP ##

isRunning = True
mousePos = [0]*2

space = pygame.image.load("space\ibg.png")
spaceX = [0]
score = 1
isDead = False
doomCount = 0

realTime = 0

baseMult = 1
timeDiff = 100
boostFuel = 1000

names = []
scores = []
megaList = []*2

multIsSaved = 0
timeSurvived = 0
isReversing = False
metMult = 5
```

```
isBoosting = False
ticker = 0
rowRange = 0


timeTillNextObstacle = 300
worldState = 0
gameOver = False


## MISC ##
titlecard = displayCard("titleCard")
title = pygame.sprite.Group(titlecard)

endcard = displayCard("endCard")
end = pygame.sprite.Group(endcard)

scorecard = displayCard("scoreCard")
nameScreen = pygame.sprite.Group(scorecard)

highscores = displayCard("highCard")
highScores = pygame.sprite.Group(highscores)

startMessage = font.render("Click Anywhere to Start" ,1,white)
```

```
# Functions and Procedures
def goExplode(x,y,reason):
    boom = explosion(x, y, "death")        # Creates an explosion at the location where
the cursor was
    boom.add(boomboom)

    boomNoise = pygame.mixer.Sound('music/kaboom.ogg')
    boomNoise.play()

def takeName(nameChosen, section):
    while nameChosen != True:
        if section == 1:
            c1 = "A"
            c1 = keyPress()
            pygame.display.flip()
            clock.tick(60)

            section = 2
        elif section == 2:
            c2 = "B"
            c2 = keyPress()
            pygame.display.flip()
            clock.tick(60)

            section = 3
        elif section == 3:
            c3 = "C"
            c3 = keyPress()
            pygame.display.flip()
            clock.tick(60)

            section = 4


        if section > 3:
            playerName = str(c1 + c2 + c3)
            return playerName



        pygame.display.flip()
        clock.tick(60)


def load_image(name):                    # For animated image loading
    image = pygame.image.load(name)
    return image

def makeMeteor(x,y):                # Creates a meteor
    meteor = obstacle("metya", x, y)
```

```
    meteor.add(entities)

def bubbleSort(inputData,ranging):
    swapcount = 0
    passes = 0
    comparisons = 0
    sorted = False

    while sorted == False:
        outerloop = (ranging - 1)
        swaps = 0

        for i in range (outerloop):
            comparisons = comparisons + 1
            if inputData[i][1] < inputData[i + 1][1]:
                temp = inputData[i]
                inputData[i] = inputData[i + 1]
                inputData[i + 1] = temp
                swaps = swaps + 1
                swapcount = swapcount + 1
        if swaps == 0:
            sorted = True
        outerloop = outerloop - 1
        passes = passes + 1

    outputData = inputData
    return outputData

def findPlayer(inpList,nameCheck):

    location = 1
    for i in range(len(inpList)):
        if nameCheck == inpList[i][0]:
            return location
        else:
            location += 1
    return location

def hyperSpace(spaceX,baseMult):

    screen.blit(space, [spaceX[0],0])
    spaceX[0] -= baseMult * 1.75                    # Increases the speed at which the
backgrounds scroll

    if spaceX[0] <= -1000:                  # Resets the backgrounds should they go
past a certain point, creating a
        spaceX[0] = 0                               # psuedo-infinite background

    return spaceX
```

```
def playMusic(state):
    pygame.mixer.music.load("music\mus" + str(state) +".ogg")

def keyPress():
    done = False
    while done == False:
        events = pygame.event.get()
        for event in events:
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_a:
                    output = "A"
                    done = True
                elif event.key == pygame.K_b:
                    output = "B"
                    done = True
                elif event.key == pygame.K_c:
                    ##print('C')
                    output = "C"
                    done = True
                elif event.key == pygame.K_d:
                    ##print('D')
                    output = "D"
                    done = True
                elif event.key == pygame.K_e:
                    ##print('E')
                    output = "E"
                    done = True
                elif event.key == pygame.K_f:
                    ##print('F')
                    output = "F"
                    done = True
                elif event.key == pygame.K_g:
                    ##print('G')
                    output = "G"
                    done = True
                elif event.key == pygame.K_h:
                    ##print('H')
                    output = "H"
                    done = True
                elif event.key == pygame.K_i:
                    ##print("I")
                    output = "I"
                    done = True
                elif event.key == pygame.K_j:
                    ###print('J')
                    output = "J"
                    done = True
                elif event.key == pygame.K_k:
                    ###print('K')
                    output = "K"
```

```
    done = True
elif event.key == pygame.K_l:
    ###print('L')
    output = "L"
    done = True
elif event.key == pygame.K_m:
    ##print("M")
    output = "M"
    done = True
elif event.key == pygame.K_n:
    ##print('N')
    output = "N"
    done = True
elif event.key == pygame.K_o:
    ##print('O')
    output = "O"
    done = True
elif event.key == pygame.K_p:
    ##print('P')
    output = "P"
    done = True
elif event.key == pygame.K_q:
    ##print('Q')
    output = "Q"
    done = True
elif event.key == pygame.K_r:
    ##print('R')
    output = "R"
    done = True
elif event.key == pygame.K_s:
    ##print('S')
    output = "S"
    done = True
elif event.key == pygame.K_t:
    ##print('T')
    output = "T"
    done = True
elif event.key == pygame.K_u:
    ##print('U')
    output = "U"
    done = True
elif event.key == pygame.K_v:
    ##print('V')
    output = "V"
    done = True
elif event.key == pygame.K_w:
    ##print('W')
    output = "W"
    done = True
elif event.key == pygame.K_x:
```

```
                ##print('X')
                output = "X"
                done = True
            elif event.key == pygame.K_y:
                ##print("Y")
                output = "Y"
                done = True
            elif event.key == pygame.K_z:
                ##print('Z')
                output = "Z"
                done = True
            else:
                output = "*"

            ##print(output)
            return output
        pygame.display.flip()
        clock.tick(60)

    pygame.display.flip()
    clock.tick(60)



def PewPewPew(x,y):
    blast = laserBolt(x, y)
    blasts.add(blast)
```

```
# -------- Main Program Loop -----------


while isRunning == True:

    if pygame.mixer.music.get_busy() == False:

        playMusic(worldState)

    events = pygame.event.get()
    for event in events:        # Check for an event
        if event.type == pygame.QUIT:       # If user clicks close window
            isRunning = False

        if event.type == pygame.MOUSEMOTION:
            mousePos[:] = list(event.pos)       # Keeps track of the cursor position when it
moves
    if mousePos[0] >= 1000:
        pygame.mouse.set_pos(999,mousePos[1])
    elif mousePos[0] <= 0:
        pygame.mouse.set_pos(0,mousePos[1])

    if mousePos[1] >= 500:
        pygame.mouse.set_pos(mousePos[0],499)
    elif mousePos[1] <= 0:
        pygame.mouse.set_pos(mousePos[0],0)

    if worldState == 1 or worldState == 2:
        pygame.mouse.set_visible(False) # Makes the mouse invisible and moves the
rocket to the cursor during game over & gameplay states
        cursor.moveIt(mousePos)
    else:
        pygame.mouse.set_visible(True)

    while worldState == 0:
        hyperSpace(spaceX,baseMult)


        realTime     = 0
        baseMult     = 1
        timeDiff     = 100
        boostFuel    = 1000
        score        = 0

        multIsSaved  = 0
        timeSurvived = 0
        isReversing  = False
        isBoosting   = False
        isDead       = False
        metMult      = 5
```

```
    ticker = 0


    for toKill in (entities.sprites()):
        pygame.sprite.Sprite.kill(toKill)



    title.draw(screen)
    screen.blit(startMessage, (350,350))


    pygame.display.flip()
    clock.tick(60)



    events = pygame.event.get()
    for event in events:
        if event.type == pygame.MOUSEBUTTONDOWN:
            worldState = 1                                  #ends title screen
            timeTillNextObstacle = 1000
            tutPassed = False
            metNumb = 0
            cooldown = 100
            cooldownrate = 2
            baseMult = 1


            pygame.mixer.music.stop()

            cursor = rocket()
            cursor.add(cursors)



if worldState == 1:

    if pygame.mouse.get_pressed()[1]:
        cooldownrate = 20

    if pygame.mouse.get_pressed()[2] and cooldown == 0:
        PewPewPew(cursor.rect.x,cursor.rect.y)
        cooldown = 100
    elif cooldown > 0:
        cooldown -= cooldownrate

    if pygame.mouse.get_pressed()[0] and boostFuel > 0:
```

```
            if multIsSaved == 0:
                saveMult = baseMult              #saves the original, pre-boost speed when
boosting
                multIsSaved = 1

            baseMult += 0.1
            boostFuel -= 4
            isBoosting = True

        elif multIsSaved == 1:
            if baseMult > saveMult + 2:                  # lowers baseMult until the speed
meets the saved copy
                baseMult -= .5
                isReversing = True
                isBoosting = False
            else:
                multIsSaved == 0
                isReversing = False


        curFuel          = font.render("Boost Fuel: " + str((round(boostFuel / 10,0))) +
"%",1,white)       # displays the current score, fuel, and time survived
        scoreDisplay = font.render("Distance Travelled: " + str(score) + "m", 1, white)
        timeDisplay   = font.render("Time Survived: " + str(timeSurvived), 1, white)

#************ Make Obstacles ************************#

        if pygame.time.get_ticks() > timeTillNextObstacle and tutPassed == True:
# Creates a new meteor if enough time has passed
            yCoord = random.randint(-50,550)                                # Sets a random y-
coordinate for the meteor
            makeMeteor(1100,yCoord)
            timeTillNextObstacle += (random.randint(500,1000))          # Increases the
value of timeTillNextObstacle by a random integer
        elif tutPassed == False and pygame.time.get_ticks() > timeTillNextObstacle:
            makeMeteor(1100,400)
            timeTillNextObstacle = 3000
            metNumb += 1
            if metNumb == 3:
                tutPassed = True
```

```
    # Update sprites

    hyperSpace(spaceX,baseMult)

    boomboom.update()
    boomboom.draw(screen)
    entities.draw(screen)
    cursors.draw(screen)
    blasts.draw(screen)

    screen.blit(curFuel,      (10 ,450))
    screen.blit(scoreDisplay, (700,450))


    if realTime > timeDiff:                  # Exponentially increases the speed of the
game as it goes on
        baseMult *= 1.01
        if boostFuel < 1000:
            boostFuel *= 1.001


    realTime += 1   # Keeps track of the time the player has survived for

    for lp in (blasts.sprites()):
        lp.moveIt(50)
        if lp.rect.x > 1050:
            pygame.sprite.Sprite.kill(lp)

    for mp in (entities.sprites()):    # Moves the meteorite particles (mp) at slightly
faster than the base multiplier
        mp.moveIt(baseMult * 1.6)
        if mp.rect.x < -100:          # removes a meteor if it goes too far off screen,
removing any slowdown
            pygame.sprite.Sprite.kill(mp)

    pygame.sprite.groupcollide(cursors, entities, True, False)   # If the cursor and a
meteor collide, the cursor will be killed
    laserHits = pygame.sprite.groupcollide(entities,  blasts, True, True)
    for hit in laserHits:
        goExplode(hit.rect.x,hit.rect.y,"explode")
        score += 200

    if len(cursors.sprites()) == 0 and isDead == False:
        isDead = True
        goExplode(cursor.rect.x - 10, cursor.rect.y - 40,"death")
        doomCount = realTime + 50
```

```
        if isDead == False:

            timeSurvived = int(round(realTime / 10,0))

            if isReversing == False:
                score += baseMult * 0.9
                score = int(round(score, 1))
            else:
                score += .25 * baseMult             # Constantly adds to the score, unless the
player is reversing or dead
                score = int(round(score, 1))
        else:
            for toKill in (entities.sprites()):             # Kills all meteors offscreen if the
player is dead
                if toKill.rect.x > 1000:
                    pygame.sprite.Sprite.kill(toKill)


        if realTime == doomCount:
            pygame.mixer.music.stop()
            worldState = 2
            doomCount = 0
        pygame.display.flip()
        clock.tick(60)

    if worldState == 2:
        hyperSpace(spaceX, baseMult)
        end.draw(screen)
        finalScore = (score + realTime)
        endMessage   = font.render("Your final score was " + str(finalScore) + " points",
1, white)
        screen.blit(endMessage, (350,350))


        events = pygame.event.get()
        for event in events:
            if event.type == pygame.KEYDOWN or event.type ==
pygame.MOUSEBUTTONDOWN:
                pygame.mixer.music.stop()
                worldState = 3
                miniCount = 0
                names = []
                scores = []
                nameMatched = False
                megaList = []

        pygame.display.flip()
        clock.tick(60)
```

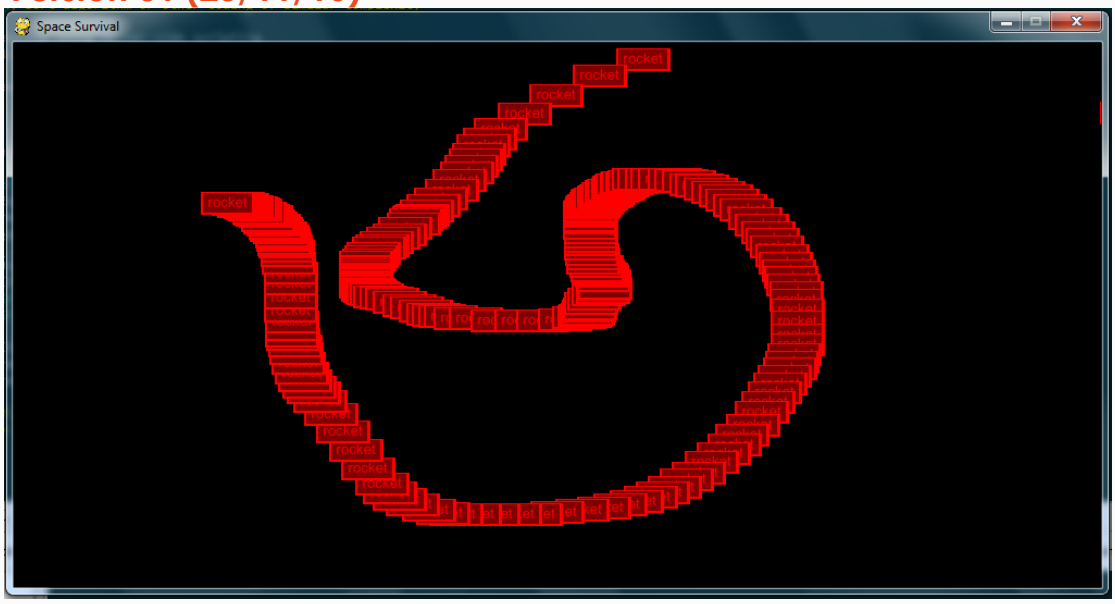
```
    while worldState == 3:

        hyperSpace(spaceX,baseMult)

        nameScreen.draw(screen)
                                                    # Input Name

        if miniCount > 10:
            playerName = takeName(False,1)
            ##print(playerName)

            with open("scoreList.csv", "r") as sl1:
                reader = csv.reader(sl1)

                for row in reader:
                    names.append(row[0])
                    scores.append(int(row[1]))

            for i in range(len(names)):
                if names[i] == playerName:
                    scores[i] = finalScore
                    nameMatched = True

            if nameMatched != True:
                names.append(playerName)
                scores.append(finalScore)

            for count in range(len(scores)):
                megaList.append([names[count],scores[count]])

            megaList = bubbleSort(megaList, len(scores))
            playPos = findPlayer(megaList, playerName)
            worldState = 4

        pygame.display.flip()
        clock.tick(60)
        miniCount += 1


    while worldState == 4:

        hyperSpace(spaceX,baseMult)

        highScores.draw(screen)

        highMessage = font.render(playerName + ", you got " + str(finalScore) + "pts!",
1, white)

        hs1  = font.render("1: " + str(megaList[0][0]) + " - " + str(megaList[0][1]) +
"pts", 3, white)
```

```
        hs2  = font.render("2: " + str(megaList[1][0]) + " - " + str(megaList[1][1]) +
"pts", 3, white)
        hs3  = font.render("3: " + str(megaList[2][0]) + " - " + str(megaList[2][1]) +
"pts", 3, white)

        if playPos == 1:
            congrats = font.render("Congratulations! You're in " + str(playPos) + "st
place!", 3, white)
        elif playPos == 2:
            congrats = font.render("Well Done! You're in " + str(playPos) + "nd place!",
3, white)
        elif playPos == 3:
            congrats = font.render("Good Job! You're in " + str(playPos) + "rd place!", 3,
white)
        else:
            congrats = font.render("You're in position " + str(playPos) + "!", 3, white)

        screen.blit(highMessage, (350,350))
        screen.blit(congrats, (300,450))

        screen.blit(hs1, (100,250))
        screen.blit(hs2, (400,250))
        screen.blit(hs3, (700,250))

        events = pygame.event.get()
        for event in events:
            if event.type == pygame.MOUSEBUTTONDOWN:

                with open("scoreList.csv","w", newline="") as sl:
                    writer = csv.writer(sl)
                    for row in range(len(scores)):
                        writer.writerow(megaList[row])


                pygame.mixer.music.stop()
                worldState = 0



        pygame.display.flip()
        clock.tick(60)



    pygame.display.flip()
    clock.tick(60)



pygame.quit()
```

Record of Progress

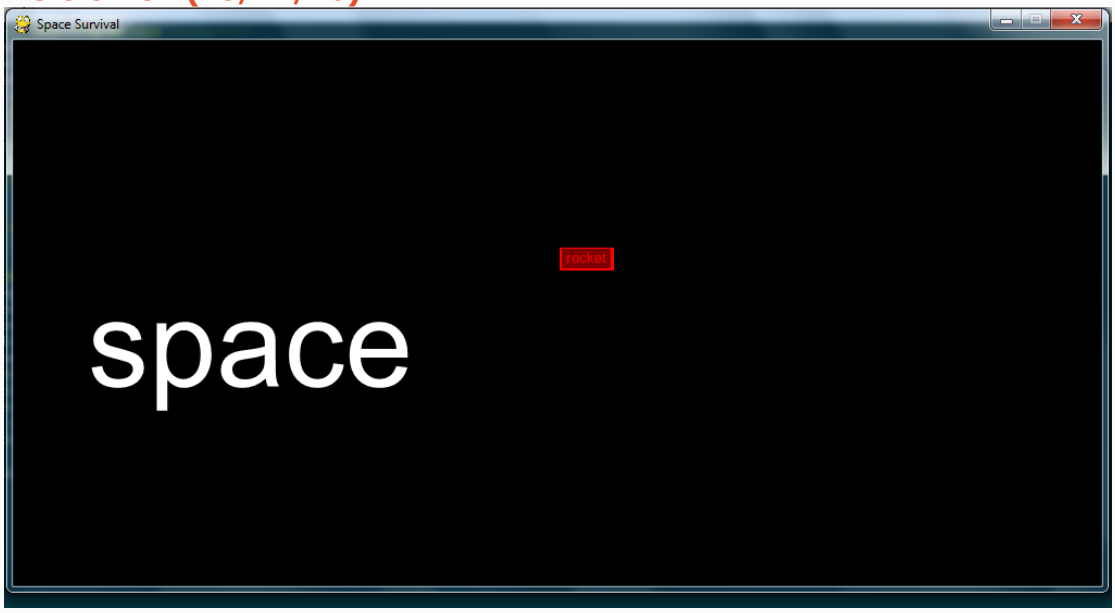
The following screenshots and brief reports have been created for the various prototypes produced during development of the game. The game itself was made using an AGILE methodology, and as such has many rough prototypes with numerous small changes that eventually produce a finished product.

Version 01 (25/11/16)



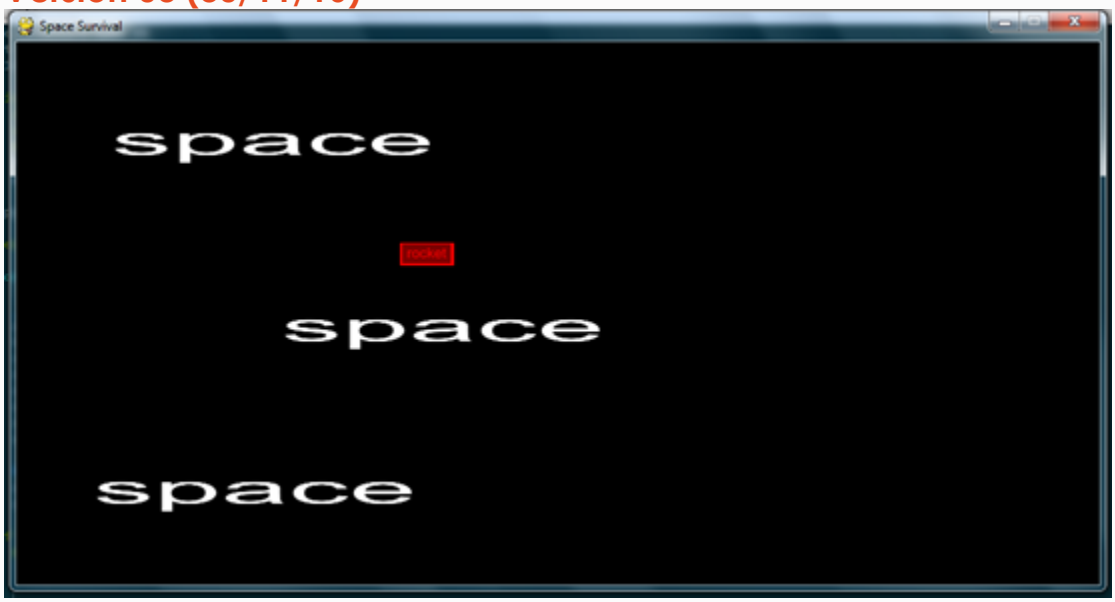
The Pygame window is being set up correctly, and a basic placeholder image has been used for the "rocket". The rocket has the coordinates on screen that the mouse cursor does, but is leaving a trail.

Version 02 (25/11/16)



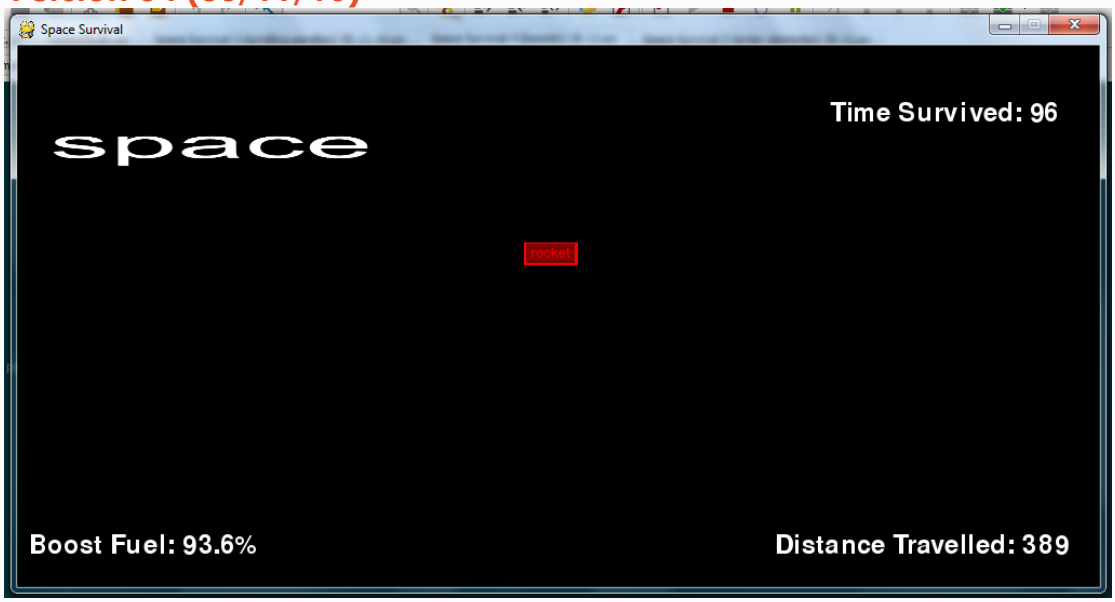
Mouse is now hidden during gameplay. A "makeMeteor" function has been added to create meteor instances (which don't exist yet), and a basic scrolling background has been implemented. The program checks if the position of the background is beyond the edge of the screen, and puts it over to the other side if it is.

Version 03 (30/11/16)



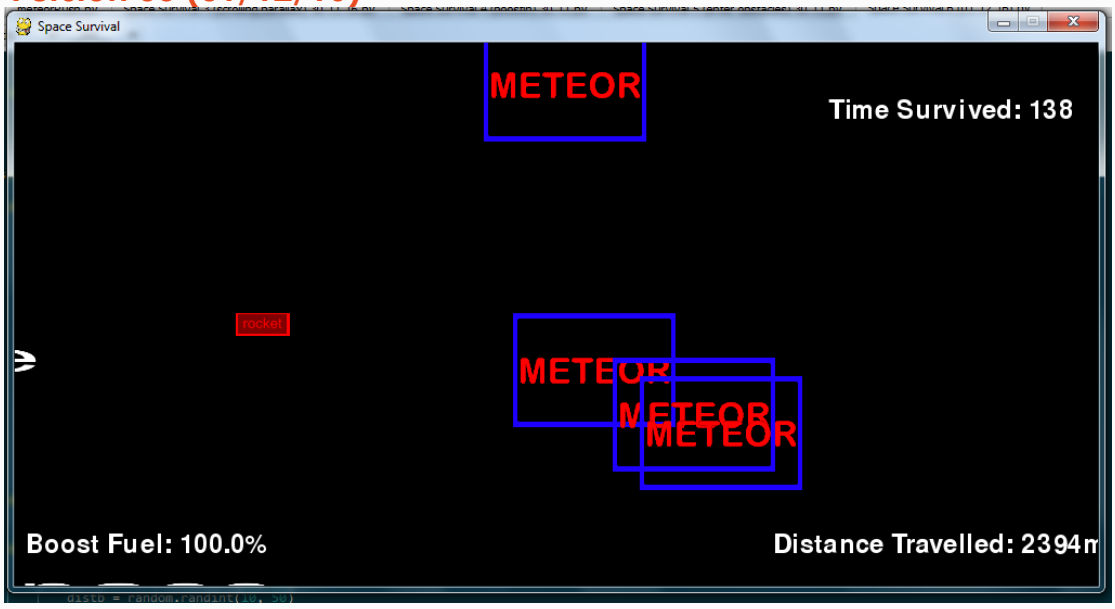
The scrolling background is now in multiple "slices", each moving at a different speed to create the illusion of depth. In addition to this, Boosting now works as intended, increasing the rate at which the backgrounds move when the player clicks the mouse.

Version 04 (30/11/16)



Counters for the score, time and boosting time remaining now render onto the screen, giving easier feedback to the player. A basic "reverse" system has been put into place, reducing the speed of the movement after boosting to just above the original speed.

Version 05 (01/12/16)



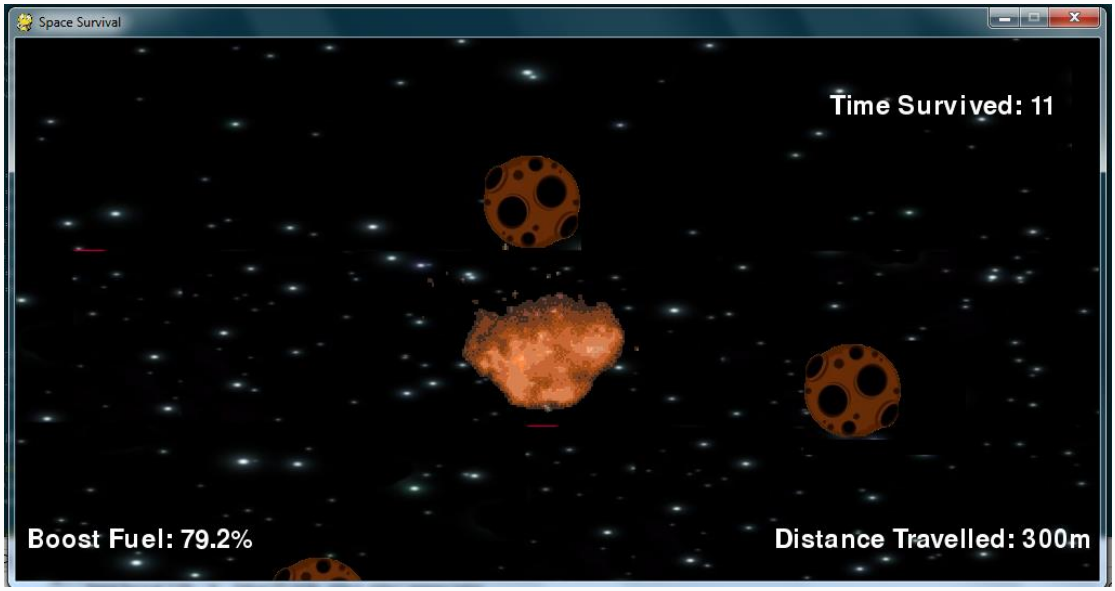
"Meteor" obstacles now spawn right of the screen and scroll left. Collision doesn't work yet, however. The graphical sources have been moved into different folders for easier organisation.

Version 06 (04/12/16)

Collision between the player and a meteor now destroys the player's "rocket", via the groupcollide function. Any meteors that go off the edge of the screen are now killed to save processing power.

Version 07 (06/12/16)

Graphics have been replaced with placeholder images, mostly from Google image searches. Hitting a meteor now creates an explosion after killing the player, using code from http://stackoverflow.com/questions/14044147/animated-sprite-from-few-images to animate itself.



Version 08 (07/12/16)

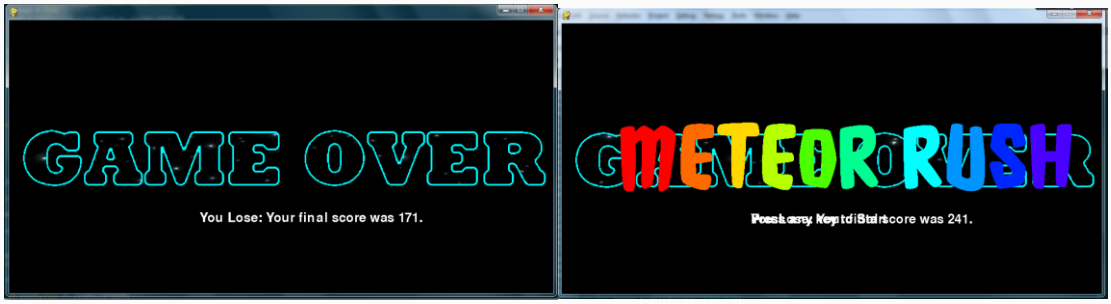
Working on implementing a title screen.

Version 09 (08/12/16)

A title screen has successfully been implemented, that will change the worldState to 1 when the user presses a button.

Version 10 (09/12/16)

A working game over screen has been implemented, as the worldState becomes 2 when the player is destroyed. A slight error here is that the game continues to draw the image even after the worldState is set back to 0, and that the game appears to not properly reset.



Version 11 (15/12/16)

Changing to a different random integer generator for the time between meteor spawns, as the current one is too slow and easy. Currently, this crashes the program. A new title screen is also being drawn up to further match the aesthetic of the program.

Version 12 (16/12/16)

The random timing now works successfully, creating significantly more meteorites. In addition to this, clicking now closes the title screen.

Version 13 (20/12/16)

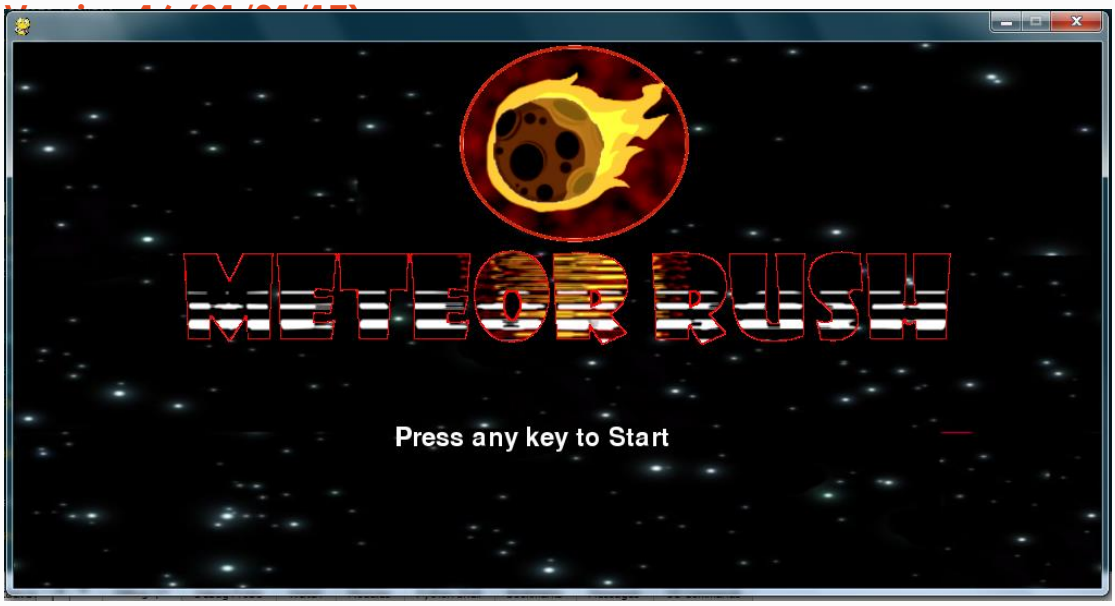
Working on a small counter to give the "explosion" animation more time to play before changing the worldstate.

Version 14 (05/01/17)

New Year! Continuing to work on both report overall and on the graphics for the other worldStates.

Version 15 (06/01/17)

Counter finally works!



Prelims now over with, so programming can continue. Changed title screen, and am currently attempting to switch the background scrolling to one that moves eight or more “slices” rather than four, for better graphical quality. Also trying “groundhog” function to reset game.

Version 17 (01/02/17)

Abandoned “groundhog” function, as it is more efficient to just manually reset the variables in play when the title screen is active than to have a function only used once. Resetting variables still needs some work, as the score does not always change.

Version 18 (02/02/17)

Reset works properly. Working on implementing the planned buttons to change worldState from the menu.

Version 19 (07/02/17)

Button work continues.

Version 20 (11/02/17)

Music is now played depending on what the current worldstate is, selected from a folder of .ogg files. The buttons are being put off indefinitely, as I am attempting to improve the framerate and backgrounds. This is difficult, as background blitting doesn’t follow the same rules as objects.

Version 21 (15/02/17)

Continuing work on space slices, currently beginning to create them as a new object to be drawn beneath all others.

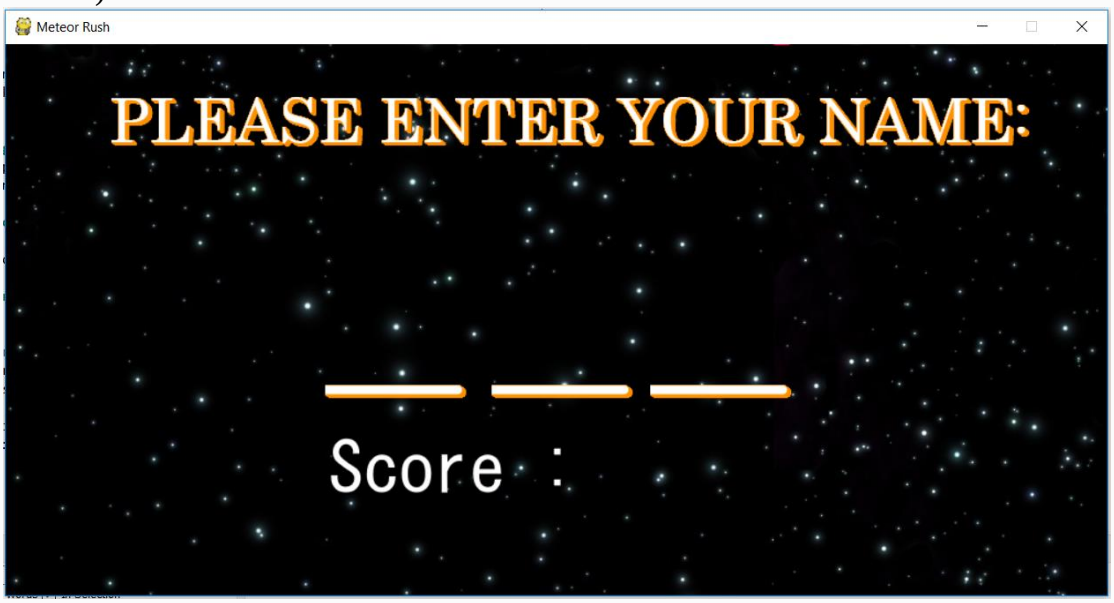
Version 22 (16/02/17)

Still working on it, beginning to think it may be a good idea to abandon the slices.

Version 23 (17/02/17)
Much the same as yesterday.

Version 24 (21/02/17)
The background has been replaced with a single scrolling image, increasing the rate at which frames are rendered massively.

Version 25 (25/02/17)
Working on the user name input screen. (Unfortunately, this file has been accidentally deleted)



Version 26 (28/02/17)
Pygame's lack of proper keyboard input is making actual input for the name screen difficult. Also continuing to attempt to solve issue on playing again after game ending where the rate at which meteors spawn increases as the player is in the main menu worldState, creating a "wall" when they play.

Version 27 (02/03/17)
Still continuing work on creating solutions for the two problems present, as well as sorting the scores in the main storage file. Polishing up finer details of earlier report as well as working on ISDD assessment, so time working on this is limited.

Version 28 (07/03/17)
Bubble sort is broken, attempting to fix it.

Version 29 (08/03/17)
Sorting function now works, but will duplicate the data taken in when writing to file. The input for the user's name is now a placeholder command prompt input until this is replaced.

Version 30 (09/03/17)
Continuing to try to find a fix for the "wall" issue. Currently, the best idea I have would be to reset the pygame clock. Trying to fix input for the name screen also.

Version 31 (10/03/17)
A workaround has been implemented for the name input, as the program will now search for key presses while on the worldState.

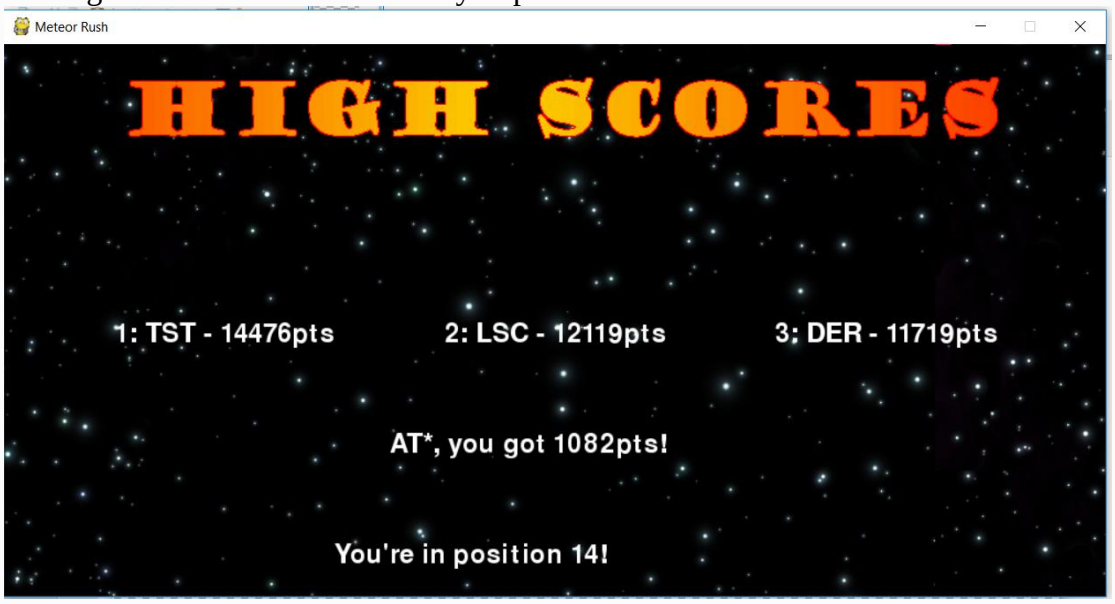
Version 32 (14/03/17)
Continuing to work on keypresses. Buttons have been abandoned, as they don't seem necessary within the confines of the program. A graphic for the high score screen has been created.

Version 33 (15/03/17)
An alternative to fixing the "wall" issue has been found, as implementing the "laser fire" function allows the player to destroy the wall rather than avoid it. The laser function has been mostly finished with a small graphic and functionality.

Version 34 (16/03/17)
The laser firing now has a longer cooldown, and I am working on adding an explosion effect on collision with a meteorite.

Version 35 (17/03/17)
An explosion will now be created on collision between laser and meteor, and an explosion sound effect now plays on creation of an explosion.

Version 36 (20/03/17)
The high score screen has been fully implemented.



Version 37 (24/03/17) (Final Version)
All planned features have either been implemented or dropped. The framerate of the game has been improved by removing filling the screen every frame, and a small system has been implemented to lock the mouse to the screen region when possible.

Testing

Subject	Input	Expected Output	Actual Output	Output Matches?
“MousePos” value used in Rocket class’s “moveIt” function	Mouse moved upwards	“cursor” graphic goes upward to mouse position, “x” value is lowered	“cursor” graphic goes upward to mouse position, “x” value is lowered	Yes
^	Mouse moved left	“cursor” graphic goes left to mouse position, “y” value is lowered	“cursor” graphic goes left to mouse position, “y” value is lowered	Yes
Bubble sort function	Csv file containing “TOP,999 LOW,0 MID,500 ESC,123”	Csv file is sorted into “TOP,999 MID,500 ESC,123 LOW,0”	Csv file is sorted into “TOP,999 MID,500 ESC,123 LOW,0”	Yes
Bubble sort function on a second iteration	Csv file containing “TOP,999 MID,500 ESC,123 LOW,0”	Csv file remains unchanged	Values in table are duplicated: “TOP,999 TOP,999 MID,500 MID,500 ESC,123 ESC,123 LOW,0 LOW,0”	No. The arrays holding the information are not being purged between games.
“Boosting” function	Player uses left click	Screen moves past at an increased rate, slows back to normal when clicking stops.	Screen moves past at an increased rate, then rapidly reverses when clicking stops.	No. The original “baseMult” value will need to be saved to reduce the speed to it instead of just lowering the variable infinitely.

Player death	Cursor moves position to that of a “meteor”	Rocket is removed, explosion graphic is placed, worldState changes to “game over” state	Rocket is removed, explosion graphic is placed, worldState changes to “game over” state	Yes
“Laser fire” function	Player clicks right mouse button	“Laser” entity is created on cursor location, begins moving right.	“Laser” entity is created on cursor location, begins moving right.	Yes
Name input	Worldstate is 4, player enters three letters to make name	playerName variable is created by concatenation of three letters, game progresses to next state.	Variable is created, but the screen freezes and if a wrong key is pressed, crashes.	No

Evaluation

The final program is suitable for the original purpose (being playable and functioning correctly), meeting all the specifications. The interface is able to convey information to the user quickly, but the lack of an in game instruction screen has caused some confusion. To avoid this, a text file with basic instructions is included. The program functions correctly after a lot of testing, although frame rates are often inconsistent.

The record of progress I have included could be a lot more detailed in actual progress made, and shows that the program could have been developed significantly faster had it been more planned out. As well as this, I neglected to keep an actual record of the hours spent on any particular section of the project. This coupled with the program going well past the intended deadline, shows that the project work needed to be a lot more tightly organised.

Overall, the program has been a success, but has shown that I must be more able to plan the end product before starting a program in the future, as well as being more focussed whilst coding the program itself.

Issues & Solutions within Project:

Issue	Solution
Pygame lacks proper keyboard input	Event checker for key presses implemented, main motion is mouse based
Spawn rates continue to escalate while in menu state, creating a “wall” of meteors upon starting a new game	Implemented “laser fire” to allow destruction of meteors, turning unpassable obstacle to challenge for players
Framerates slow	Removed multiple scrolling backgrounds, screen region filling to lower resource consumption
Lack of help screen	Readme text file created
Scores duplicating	Clearing arrays between games entirely, fully rewriting the csv file when writing to file